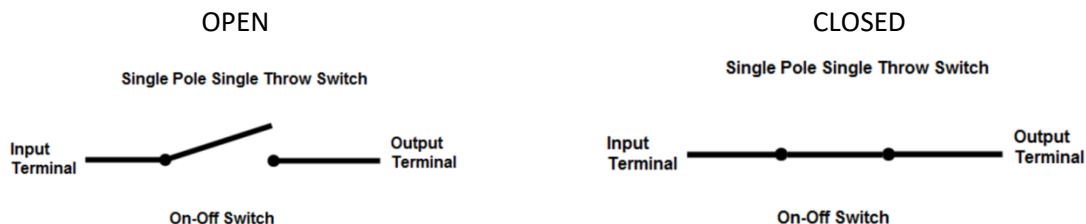# EdgeDebounce V1.2 (Released August 23, 2017)

TUTORIAL

SWITCHES ARE ON OR OFF RIGHT?

Yes and no. Switches are analog devices. They are designed to let current flow thru them. Usually, you have two conductive materials that come into contact after the user has done something to bring them together or to take them apart. When the conductors are apart, there is no current flowing and the switch is off (the switch is open). When the conductors are in contact, the current flows and the switch is on (the switch is closed). So most of the time, the switch is off or on.

OPEN   CLOSED

Single Pole Single Throw Switch   Single Pole Single Throw Switch

Input Terminal   Output Terminal   Input Terminal   Output Terminal

On-Off Switch   On-Off Switch

But many things can happen when there is a transition between off and on. You have seen thunderstorms right? This phenomenon can happen in a switch. Given sufficient humidity, sufficient particles in suspension, there can be "lightning" happening inside your switch when it is pressed or released. The contact has not been made (or released) yet, but there is current flowing. Rest assured nothing there to send a DeLorean into the future. None the less, if you ask your Arduino to report the switches' status at the exact time a "lightning" occurs in your switch, you will get a false reading. All that your Arduino can tell you when you only ask once what is the state of my switch right now is… that's how it looks right now.

Here is an example of a thunderstorm[1]:
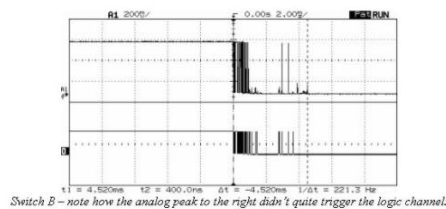
Switch B – note how the analog peak to the right didn't quite trigger the logic channel.

On the top, you can see the real voltages that went thru the switch. On the bottom, you see what an analog to digital converter (like digitalRead()) does to this signal. When the switch was released and the contacts where still really close, current went thru erratic states. The time between a full on to a full off state is around 2ms for that switch. That is 2 thousands of a second. Your Arduino can read a pin around 650 times during this period. During that brief period, what digitalRead() reports is likely wrong.

Some switches are smoother than others when going from low to high or vice-versa. Here is an example:
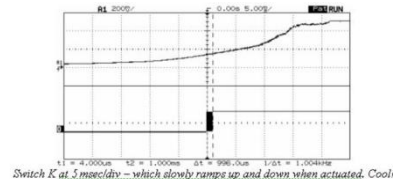
Switch K at 5 msec/div – which slowly ramps up and down when actuated. Cool!

That could be cool, but the problem here, is how Arduino decides if a signal is high or is low. The specs are clear: A TTL input signal is defined as "low" when between 0 V and 0.8 V with respect to the ground terminal, and "high" when between 2.2 V and 5 V. In between, Arduino will report low or high in a random manner, and will not let you know that it is doing so. Looking at the diagram, you can see that during the period that the analog to digital is uncertain, (between 0.8 and 2.2 V) anything goes.

---

[1] All examples are taken from : http://www.ganssle.com/debouncing.htm

HOW CAN I BE SURE THAT A SWITCH IS ON OR OFF

As we saw earlier, bouncing only happens when going from high to low or vice-versa. Our approach will be to be patient. We can read successively many times the switch and decide if it is on or off. Let's say that instead of reading the switch once, we read it 16 times in a row, as fast as Arduino can (That takes around 90 microseconds). If the switch is off, the 16 reads will return the same value. The same goes when the switch is on. The problem only occurs when there is a transition from off to on and vice-versa. The obvious way to detect this is to make sure that there is no discrepancy between successive reads of the pin. If all consecutive reads report high, we can be reasonably certain that the pin is high. The same goes with lows. Otherwise, if some digitalRead() report high and some report low, we are certain that we are in a transition phase. As said before, we need to be patient. We will read another batch of 16 digitalRead() again and again until the signal is stable (16 high or 16 low).

THE ALGORITHM[2]
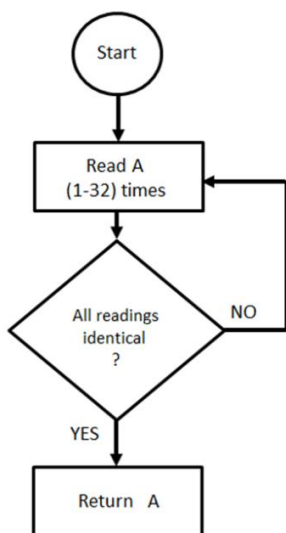
1. Read the pins consecutively a certain amount of times (the Library can be set to read between 1 (no debouncing) and 32 times maximum debouncing).
2. If all the reads are the same (all low or all high), return that value.
3. If not, return to step 1

By default, the number of reads is set to 16. You can use the "TestSensitivity.ino" sketch to play around with this value and see what is the best choice for your particular switch or application.

CONNECTING THE SWITCH

There are two ways to connect a switch:
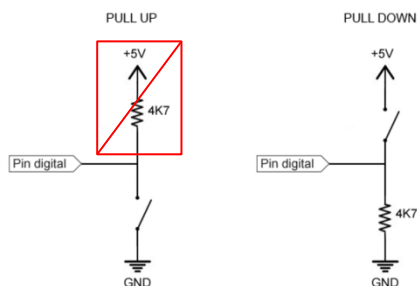
In PULL UP mode, on the left, one of the switches' pins is connected to Arduino's GND pin and the other is connected to a digital pin (D2..D13 on an UNO). You don't have to place the resistor that is between +5V and the digital pin. Arduino has its own internal pull up resistor.

In a sketch, with the switch connected in pull up mode, you would, in the setup() section, write this line:

```
pinMode(2, INPUT_PULLUP)
```

When in pull up mode, when the switch is closed (ON), digitalRead(2) returns LOW, and when the switch is open (OFF), digitalRead(2) returns HIGH

---

[2] This algorithm is based on the recommendations on this page: http://www.ganssle.com/debouncing-pt2.htm

In PULL DOWN mode, on the right, one of the switches' pins is connected to Arduino's +5V pin and the other is connected to a digital pin (D2..D13 on an UNO). You **have to** place a 4K7 ohm resistor between the digital pin and Arduinos' GND pin. This makes sure that the digital pin is LOW when the switch is open.

In a sketch, with the switch connected in pull up mode, you would, in the setup() section, write this line:
```
pinMode(2, INPUT)
```

When in pull down mode, when the switch is closed (ON), digitalRead(2) returns HIGH, and when the switch is open (OFF), digitalRead(2) returns LOW

PULLDOWN		PULLUP	
OPEN	CLOSED	CLOSED	OPEN
LOW	HIGH	HIGH	LOW

THE LIBRARY

SETTING UP THE SWITCH

To use the Library, you have to install it in Arduino's IDE. You will find a good tutorial to do it at this address: https://www.arduino.cc/en/Guide/Libraries

In your sketch, you have to mention that you want to use the library. This is done with this line of code which has to be near the top of your sketch, before the setup() section.
```
#include <EdgeDebounce.h>
```

Right after that, you can give a name to the pin that is attached to your switch (You don't have to, but it is good practice to do so).
```
#define BUTTON_PIN 2
```

Next, you have to instantiate your switch. It creates an object of a class Debounce and gives it the name that you choose. The debounce class needs to know two things in order to create the object:
First, it needs to know on which Arduinos' pin you connected the switch.
Next, it needs to know how you connected the switch: PULLUP or PULLDOWN.

The following line creates an Edge**Debounce** object named **button,** connected to the pin **BUTTON_PIN** that has been wired in **PULLUP** mode.
```
EdgeDebounce button(BUTTON_PIN, PULLUP);
```

In the setup() section of your code, you don't have to place the usual pinMode() line. Just use :
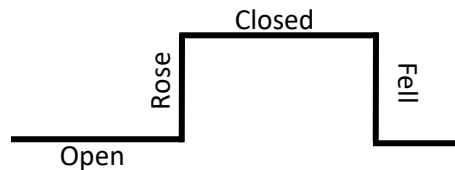```
button.begin();
```

You can ask if the switch is **pressed**. Remember:

- In PULLUP mode, pressed means HIGH if open and LOW if closed;
- In PULLDOWN mode, pressed means LOW if open and HIGH if closed;

```
if (button.pressed() == HIGH) {
  //Your code here;
}
```

Version 1.2: this method has been renamed **debounce**. We can still use .pressed()

```
if (button.debounce() == HIGH) {
  //Your code here;
}
```

Closed

Rose

Fell

Open

When a switch is open, it does not conduct current. When it is closed, it conducts current. At the moment that the switch is closed, we say that the signal is rising. At the moment that it is opened, we say that the signal is falling.

Version 1.1: We can ask if the switch is **closed**:

```
if (button.closed()) {
  Your code here;
}
```

Version 1.2: To know if it is **open**, we can ask:

```
if (button.open()) {
  Your code here;
}
```

Version 1.2: We now can also find out if:

It just **rose.**

```
if (button.rose()) {
  Your code here;
}
```

It just **fell.**

```
if (button.fell()) {
  Your code here;
}
```

NOTE: If a switch is responding to a rising or falling edge of the signal, it has to concentrate on only one task. Calling any other method will certainly make it miss the edge. When the switch is pressed then released, there is only one out of the hundred thousand times that the method is called where the switch will actually be rising. The same goes for the falling edge.

If you really need to have the switch check for **both** the rising **and** the falling edge, do this:

```
button.update();
if (button.getRose()) {
  //Your code here
}
if (button.getFell()) {
  //Your code here
}
```

One last thing that the EdgeDebounce object can do is to set its sensitivity between 1 and 32 (defaults to 16).

In the setup() section of your sketch, you can set it with:
```
button.setSensitivity(8);
```

In the setup() section of your sketch, you can find out its value with:
```
byte currentSensitivity = button.getSensitivity();
```

ONE OF THE USEFULL WAYS TO CALL .fell()

If we want to react on a click (button pressed and released) the classical way to do this is:
```
if (digitalRead(pin, HIGH) {
  while (digitalRead(pin, HIGH) {;} //Wait
  //Your code here
}
```

With the fell() method:
```
if (clicker.fell()) {   //"clicker" can only ask .fell()
  //Your code here
}
```

COMPLETE EXAMPLE

```
#include <EdgeDebounce.h>

#define BUTTON_PIN 2
#define LED_PIN 13

//Create an instance of EdgeDebounce and name it button
//button is tied to pin BUTTON_PIN and is in PULLUP mode
EdgeDebounce button(BUTTON_PIN, PULLUP);

void setup() {
  button.begin();
  pinMode(LED_PIN, OUTPUT);
}

//Debug LED on pin 13 will light up when the button is closed (conducting)
void loop() {
  if (button.closed()) digitalWrite(LED_PIN, HIGH);
  else                 digitalWrite(LED_PIN, LOW);
}
```

I sincerely hope that this EdgeDebounce Library will help you in your projects.
Jacques Bellavance